AMENDMENTS TO THE CLAIMS

1. (Currently Amended) A method in a computer system for placing a task in a known state, the task having multiple threads executing on streams of a processor of the computer system, the processor having multiple streams for simultaneously executing threads of the task, the method comprising:

notifying each of the threads of the task executing on a stream of the processor to enter a known state; and

for each of the threads,

in response to receiving the notification, entering the known state so that an action can be performed with the task being in the known state.

2. (Original) The method of claim 1 wherein the known state is a quiescent state.

3. (Original) The method of claim 1 wherein the known state is where each of the threads is executing idle instructions.

4. (Original) The method of claim 1 wherein the known state is where the threads stop executing instructions.

5. (Original) The method of claim 1 wherein the task is assigned to a protection domain and the notifying includes raising a domain signal for the protection domain.

6. (Original) The method of claim 1 wherein prior to entering the known state each thread saves its state.

7. (Original) The method of claim 1 wherein a thread of the task initiates the notifying.

8. (Original) The method of claim 7 wherein the thread initiates the notifying by sending a request to an operating system.

9. (Original) The method of claim 7 wherein the task notifies an operating system that the task is blocked from further productive use of a resource until an event occurs prior to entering the known state.

10. (Original) The method of claim 1 wherein the action to be performed is swapping out the task from processor utilization.

11. (Original) The method of claim 10 wherein prior to entering the known state each thread saves its state information.

12. (Original) The method of claim 1 wherein each thread of the task has state information and the action to be performed is review of the state information.

13. (Original) The method of claim 12 wherein the review of the state information is by a debugger.

14. (Original) The method of claim 13 wherein the debugger executes as a thread of the task that does not enter the known state.

15. (Original) The method of claim 1 wherein the action is to process a signal indicated by an operating system.

16. (Original) The method of claim 1 wherein the action is to perform an inter-thread long jump.

17. (Original) The method of claim 1 wherein the known state is waiting on a synchronization indication.

18. (Original) The method of claim 17 wherein the waiting is performed by accessing a memory location with a synchronization access mode of future.

19. (Original) The method of claim 1 wherein entering the known state includes invoking an operating system call.

20. (Original) The method of claim 1 wherein a master thread is designated for notifying an operating system when the task is in the known state.

21. (Original) The method of claim 1 wherein one of the threads enters a known state of processing signals while the other threads are in a known state that is a quiescent state.

22. (Original) The method of claim 21 wherein after processing the signals each of the threads exits the known state.

23. (Currently Amended) A method in a computer system for a task to exit a known state, the computer system supporting multiple streams, the task having multiple threads executing on the streams, the method comprising:

notifying each of the threads of the task executing on a parallel processor architecture having multiple simultaneously executing protection domains to exit the known state; and

for each thread,

in response to receiving the notification, executing instructions that were to be executed prior to entering the known state.

24. (Original) The method of claim 23 wherein the known state is a quiescent state.

25. (Original) The method of claim 23 wherein the known state is where the threads are executing idle instructions.

26. (Original) The method of claim 23 wherein the known state is where the threads stop executing instructions.

27. (Original) The method of claim 23 wherein the known state is where the threads are waiting on a synchronization indication.

28. (Original) The method of claim 27 wherein the notifying includes indicating the synchronization.

29. (Original) The method of claim 23 wherein each thread restores state information that was saved prior to the state entering the known state.

30. (Original) The method of claim 29 wherein one thread performs signal processing upon exiting the known state.

31. (Original) The method of claim 30 wherein the other threads wait until the signals are processed before executing instructions that were to be executed prior to entering the known state.

32. (Original) The method of claim 23 including reserving a number of streams for the task.

33. (Original) The method of claim 23 wherein after receiving the notification, the task creates streams for the threads.

34. (Currently Amended) A system for placing a task in a known state, the task having multiple threads executing on streams of the system, the system comprising:

a component for notifying each of the threads of the task executing on a parallel processor architecture having multiple simultaneously executing streams to enter a known state; and

a component for each of the threads that,

in response to receiving the notification, causes the thread to enter the known state so that an action can be performed with the task being in the known state.

35. (Original) The system of claim 34 wherein the known state is a quiescent state.

36. (Original) The system of claim 34 wherein the known state is where each of the threads is executing idle instructions.

37. (Original) The system of claim 34 wherein the known state is where the threads stop executing instructions.

38. (Original) The system of claim 34 wherein the task is assigned to a protection domain and the notifying includes raising a domain signal for the protection domain.

39. (Original) The system of claim 34 wherein prior to entering the known state each thread saves its state.

40. (Original) The system of claim 36 wherein a thread of the task initiates the notifying.

41. (Original) The system of claim 40 wherein the thread initiates the notifying by sending a request to an operating system.

42. (Original) The system of claim 40 wherein the task notifies an operating system that the task is blocked from further productive use of a resource until an event occurs prior to entering the known state.

43. (Original) The system of claim 34 wherein the action to be performed is swapping out the task from processor utilization.

44. (Original) The system of claim 43 wherein prior to entering the known state each thread saves its state information.

45. (Original) The system of claim 34 wherein each thread of the task has state information and the action to be performed is review of the state information.

46. (Original) The system of claim 45 wherein the review of the state information is by a debugger.

47. (Original) The system of claim 46 wherein the debugger executes as a thread of the task that does not enter the known state.

48. (Original) The system of claim 34 wherein the action is to process a signal indicated by an operating system.

49. (Original) The system of claim 34 wherein the action is to perform an inter-thread long jump.

50. (Original) The system of claim 34 wherein the known state is waiting on a synchronization indication.

51. (Original) The system of claim 50 wherein the waiting is performed by accessing a memory location with a synchronization access mode of future.

52. (Original) The system of claim 34 wherein entering the known state includes invoking an operating system call.

53. (Original) The system of claim 34 wherein a master thread is designated for notifying an operating system when the task is in the known state.

54. (Original) The system of claim 34 wherein one of the threads enters a known state of processing signals while the other threads are in a known state that is a quiescent state.

55. (Original) The system of claim 34 wherein after processing the signals each of the threads exits the known state.

56. (Currently Amended) A system for use by a task to exit a known state, the system supporting multiple streams, the task having multiple threads executing on the streams, the system comprising:

a component for notifying each of the threads of the task to exit the known state, wherein the task is executing on a parallel processor architecture having multiple simultaneously executing protection domains; and

a component for each thread that,

in response to receiving the notification, executes instructions that were to be executed prior to entering the known state.

57. (Original) The system of claim 56 wherein the known state is a quiescent state.

58. (Original) The system of claim 56 wherein the known state is where the threads are executing idle instructions.

59. (Original) The system of claim 56 wherein the known state is where the threads stop executing instructions.

60. (Original) The system of claim 56 wherein the known state is where the threads are waiting on a synchronization indication.

61. (Original) The system of claim 60 wherein the notifying includes indicating the synchronization.

62. (Original) The system of claim 56 wherein each thread restores state information that was saved prior to the state entering the known state.

63. (Original) The system of claim 62 wherein one thread performs signal processing upon exiting the known state.

64. (Original) The system of claim 63 wherein the other threads wait until the signals are processed before executing instructions that were to be executed prior to entering the known state.

65. (Original) The system of claim 56 including reserving a number of streams for the task.

66. (Original) The system of claim 56 wherein after receiving the notification, the task creates streams for the threads.